02

Creando conexión

Transcripción

Codigo:

```
public class Conexion {

    public Conexion() {
        System.out.println("Abriendo conexion");
    }

    public void leerDatos() {
        System.out.println("Recibiendo datos");
        // throw new IllegalStateException();
    }

    public void cerrar() {
        System.out.println("Cerrando conexion");
    }
}
```

COPIA EL CÓDIGO

[00:00] ¿Qué tal? Bienvenidos a la última clase del módulo de Excepciones de su curso en formación de Java. Primero, felicidades por haber llegado hasta este punto. Excepciones no es un módulo fácil, pero con estudio y práctica, sobre

todo práctica, van a poder agarrar el truco para conseguir manejar errores en su código.

[00:29] Hay un par de cosas más aquí que me gustaría enseñarles. Para esto yo voy a cerrar nuevamente todas mis clases aquí. A mí me gusta bastante tener mi espacio de trabajo limpio, no tener varias clases abiertas sino abro solamente con lo que yo necesito trabajar. Y ya no voy a usar más el proyecto herencia y polimorfismo.

[00:55] Es por eso que no es necesario que lo descarguen para este curso, yo lo he usado aquí para fines de ejemplo pero no va a haber alguna modificación extra para los cursos futuros. Y voy a volver aquí a mi proyecto de pila de ejecución. ¿Qué es lo que voy a hacer aquí? Ahora vamos a olvidarnos solo un momentito de excepciones y vamos a mencionar un concepto de aplicaciones distribuidas.

[01:27] ¿Qué son las aplicaciones distribuidas? Aplicaciones distribuidas es un conjunto de aplicaciones que interactúan entre sí para conseguir satisfacer las necesidades de negocio de una misma empresa, por ejemplo. Para tomar un ejemplo de la vida real y sea más fácil de entender este concepto, vamos a hablar de Facebook.

[01:56] Facebook es un ejemplo ideal de aplicación distribuida. ¿Por qué? Porque toda la funcionalidad que tiene Facebook no está en un solo proyecto. Está en varios proyectos y estos proyectos a la vez se comunican entre sí para conseguir satisfacer todas las reglas de negocio que tiene Facebook. Por ejemplo, manejar listas de amigos, manejar grupos, modificaciones de contactos, ubicación en tiempo real, subir tus fotos, todo eso no está en un solo sistema, está en varios sistemas.

[02:34] Está distribuido en varios sistemas y estos sistemas interactúan entre sí para conseguir satisfacer todas esas necesidades de negocio. Entonces, este es

el concepto macro, en grande. Vamos ahora a un punto un poco más pequeño. Sabemos pues que las aplicaciones tienen lo que es también la base de datos.

[03:00] ¿Qué es la base de datos? Es el repositorio en el cual pues todas las transacciones que va haciendo la aplicación van a ser pues persistidas, guardadas y van a quedar pues para que en el futuro pueda ser accesible y podamos obtener esos datos.

[03:21] Por ejemplo si yo necesito tener un histórico de saques de un usuario en mi Byte Bank, lo necesitaría, pues una base de datos para guardar todo ese histórico de cuántas veces él llamó al método saca, cuánto monto sacó, cuánto monto depositó, todo eso va en la base de datos. Pero no vamos a profundizar tanto ahí porque eso ya va a ser parte de un módulo futuro de su curso de formación en Java.

[03:47] Solo que es necesario hablar de esto para dar un mejor entendimiento a la siguiente idea que les quiero explicar. Lo que yo voy a hacer aquí es crear una clase, y esa clase se va a llamar conexión. Así. ¿Y cuál es la idea de crear esta clase conexión? Yo aquí voy a simular una conexión con una base de datos entonces para eso ya tengo un código preparado. Lo que voy a hacer es copiarlo.

[04:25] Ese código, ustedes lo van a encontrar aquí abajo, también, detallado para que también puedan copiarlo. ¿Y qué está haciendo este código? Vamos a ver aquí. Antes que nada, él está creando, pues un constructor, él define un constructor y adentro del constructor, ¿qué dice? Abriendo conexión. Entonces, con esto yo, ¿qué quiero dar a entender?

[04:53] Con esta clase, yo le estoy diciendo que estoy abriendo en la conexión hacia una base de datos, por ejemplo. Aquí tengo un método void, leerDatos, con el cual yo voy a obtener los datos de la base de datos y pues bueno, esto es solo digamos para darle contexto, no voy a implementar este método para

obtener los datos, hacer una conexión real, pero es para tener la idea de cómo debería funcionar.

[05:19] Finalmente tengo el método cerrar, en el cual yo estoy diciendo simplemente aquí estoy cerrando la conexión. Perfecto, entonces, nuevamente recapitulando, aquí abro la conexión, cuando yo la creo abro una conexión, recibo los datos, cierro la conexión. Esa es la la idea central de esta clase. Y ahora vemos que aquí hay una excepción comentada.

[05:52] Vamos a ver porque en un momento. Yo aquí voy a crear nuevamente un test. Y se va a llamar, ¿adivinen cómo? TestConexion. Le damos finish y aqu lo que yo voy a hacer es crear una nueva conexión. Conexión con = new Conexion. Perfecto. Entonces guardamos aquí. Y yo me olvidé aquí algo muy importante.

[06:27] Si ustedes han estado atentos a todas las clases desde la primera, se van a dar cuenta. Yo estoy seguro que llegaron a ese mismo punto. Ahora sí. Me había olvidado de declarar el método main. Entoncesno iba a poder conseguir crear esa conexión. Ahora que he declarado mi conexión, si yo llamo aquí a conexión.leerDatos, él va a leer los datos. Si llamo aquí a con.cerrar, va a cerra la conexión. ¿Están seguros? Vamos a probar.

[07:04] Y en efecto él abre la conexión, recibe los datos, cierra la conexión. Ahora, ¿qué pasaría si yo descomento aquí esta línea de aquí? Throw new IllegalStateException. Este tipo de excepción es muy común cuando yo tengo algún tipo de error a nivel ya de comunicación, de conexión con otros servicios, ya sea banco de datos o sea, pues un servicio distribuido, otro servicio u otra aplicación puede ser.

[07:40] Entonces IllegalStateException es cuando simplemente la conexión no fue exitosa y dio algún tipo de error. Y yo aquí cuando estoy leyendo los datos, estoy lanzando IllegalStateException, perfecto. Entonces voy a guardar aquí y

voy a repetir el test aquí y cuando él recibe los datos, vemos aquí claramente que él dio un IllegalStateException. Perfecto.

[08:11] Entonces, nosotros ya sabemos cómo tratar con excepciones. ¿Qué es lo que voy a hacer yo? Usar un try/catch. ¿Entonces qué voy a hacer aquí? Voy a llamar a try y le voy a decir esto: "Dentro del try, vas a leer los datos y vas a cerrar la conexión." Ahora catch: ¿Qué excepción vas a atrapar? Vas a atrapar esta excepción de aquí: IllegalStateException". Perfecto.

[08:46] Y me vas a imprimir el StackTrace de esa excepción, printStackTrace. Entonces, si ejecutamos nuevamente, vamos a guardar, si ejecutamos nuevamente vemos que bueno, él recibió los datos y lanzó un IllegalStateException y imprimió el StackTrace. Para asegurarnos que entró en el catch, vamos a imprimir aquí "Recibiendo exception". Perfecto.

[09:19] Volvemos a probar y, en efecto, conexión llegó aquí, leyó los datos, explotó la bomba, cayó en el catch, catch recibió la excepción y nos pintó la traza del error, el StackTrace. Y hasta ahora está según todo lo que hemos venido aprendiendo hasta ahora, hemos hecho un buen tratamiento de la excepción, pero ahora llega una cosa muy importante.

[09:52] Nosotros hemos abierto una conexión aquí, hemos recibido los datos y hemos recibido la excepción. ¿Qué pasó con la conexión? Quedó abierta. Y cuando ya trabajamos con aplicaciones reales, sean con bases de datos u otras aplicaciones, la conexión siempre tiene que cerrarse, no podemos dejar múltiples conexiones abiertas porque después eso va a generar otro tipo de errores que ya vamos a ver en otros cursos.

[10:33] Pero la idea que quiero transmitir ahora es que toda conexión que abrimos, tenemos que cerrarla siempre, siempre deberíamos llamar el método cerrar, para asegurarnos que estamos cerrando la conexión.

[10:46] Y en efecto yo estoy llamando aquí el método cerrar, pero si vamos a la pila de ejecución, a cómo esto está desencadenándose, como yo estoy lanzando

el error aquí a este nivel, él va a llegar al catch y este código jamás va a ser ejecutado. ¿Qué podría hacer yo para cerrar la conexión?

[11:17] Bueno, la primera solución que se me puede ocurrir es de repente cerrar la conexión aquí abajo en el catch. Puedo hacer esto, puedo cortar aquí y después de imprimir el StackTrace, cierro la conexión, perfecto. Pruebo nuevamente y muy bien. ¿Qué hice aquí? Abrí la conexión, recibí misión datos, recibí MiException y cerré la conexión. ¿Qué pasaría si nuevamente yo comento esta línea?

[11:54] La conexión fue exitosa, recibí los datos exitosamente. Voy a abrir aquí y abre conexión, recibe datos y no cierra la conexión. Aquí ya la cosa comienza pues a ponerse un poco más complicada. ¿Por qué? Porque entonces yo necesitaría cerrar aquí y cerrarlo aquí, asegurarme ya, si eres exitoso, cierro la conexión aquí. Si la llamada no es exitosa, entonces también cierro en el catch

[12:29] ¿Puede resolver? Vamos a probarlo. Resuelve. Cerró aquí, vamos con el caso contrario, supongamos que hubo el error, probamos aquí, cerró la conexión aquí. ¿Solucionamos el problema? Sí, solucionamos el problema. Pero nuevamente quiero que se fijen aquí en estas dos líneas, aquí y aquí. Estamos repitiendo el mismo código en ambos lugares.

[13:00] Si yo dejara de escribir ese código en este lado o en este lado de aquí, m aplicación estaría simplemente insegura, ya no conseguiría tener este control de errores. Entonces hay una dependencia muy fuerte aquí. En pocas palabras yo siempre necesito llamar al método cerrar. ¿Será que hay alguna otra forma de tener este código más bonito, más resumido, más legible? Vamos a verlo en